On an Integer Multicommodity Flow Problem from the Airplane Industry*†

Bram Verweij[1], Karen Aardal[1], and Goos Kant[2]

[1] Department of Computer Science, Utrecht University,
P.O. Box 80.089
3508 TB Utrecht, the Netherlands
{bram,aardal}@cs.ruu.nl

[2] ORTEC Consultants bv,
Groningenweg 6-33
2803 PV Gouda, the Netherlands
gkant@ortec.nl

Abstract

Here we discuss a new integer multicommodity flow problem in which the commodities can not be shipped independently. The problem emerges in the routing of airplane parts from production sites to assembly sites. The parts are of such size that they have to be carried on dedicated trailers. Each type of part has its own type of trailer. A part is loaded on its trailer after it is produced, carried on its trailer to its assembly site, and then the trailer has to be recycled. The transport of the parts is done with huge specially built transportation aircrafts. For reasons of stability such aircrafts can only carry some pre-specified combinations of parts on trailers and empty trailers. We consider the problem of finding a feasible transportation plan that minimises the total flying time of the transportation aircrafts. For this purpose we develop both optimisation and approximation algorithms.

1 Introduction

In this paper we discuss a routing problem from the airplane industry, where the production and assemblage of airplane parts takes place at geographically distinct sites, and where the parts are transported on dedicated trailers in special transportation aircrafts. The model we discuss allows us to mathematically estimate the total cost involved in transporting airplane parts from production sites to assembly sites. More precisely, we aim to calculate a minimum cost integer circulation of transportation aircrafts that supports an integer circulation of trailers, that in turn supports an integer flow of the airplane parts that have to be transported, subject to loading restrictions.

*This research was (partially) supported by ESPRIT Long Term Research Project 20244 (project ALCOM IT: *Algorithms and Complexity in Information Technology*).

†This work has been done in cooperation with ORTEC Consultants b.v., Gouda, The Netherlands.

Problem description. We start by introducing some notation (see Figure 1 for an illustration). We are given a set $K = \{k_0, \ldots, k_{d-1}\}$ of commodities and a set V of sites (with $n = |V|$). The set K contains one special *plane object* (denoted by k_0) representing the transportation airplane, and *parts* and *trailers* representing the airplane parts that have to be transported and the trailers on which these parts are shipped. The trailer type corresponding to part k is denoted trailer(k). We will use parts(k) to denote the set of part types that can be carried on a trailer of type k. A *loading configuration* ℓ is a multiset with elements from K such that for each part $k \in \ell$ we have that trailer(k) occurs at least as often in ℓ as k does (each part has to be supported by a trailer). We define the characteristic vector of a loading configuration ℓ as $\chi^\ell \in \mathbb{N}^d : \chi^\ell(k) = |\{k \in \ell\}|, \forall k \in K$. So, χ^ℓ has a component $\chi^\ell(k)$ for each $k \in K$ indicating the number of times k occurs in ℓ. For any loading configuration ℓ, $\chi^\ell(k_0) = 1$. We are given a set L of loading configurations, and we say that loading configuration ℓ is *admissible* if $\ell \in L$. The set of admissible loading configurations models the different possibilities of loading a transportation aircraft with large airplane parts. This has to be done in such a way that the capacity restrictions are satisfied, and such that the transportation aircraft remains stable during the flight. L is the complete set of loading configurations that satisfy these constraints. Our algorithms require one more technical assumption on L, namely that a special set of so-called *preferred loading configurations* is contained in L. Preferred loading configurations are defined in Section 2.2. The production and demand of the sites for each commodity is given in the matrix $B \in \mathbb{Z}^{n \times d}$. We denote element (i, k) of B by b_{ik} and the i^{th} row of B by $\boldsymbol{b}_i$. Element $b_{ik} > 0$ if site i produces commodity k, $b_{ik} < 0$ if site i assembles commodity k, and $b_{ik} = 0$ otherwise.

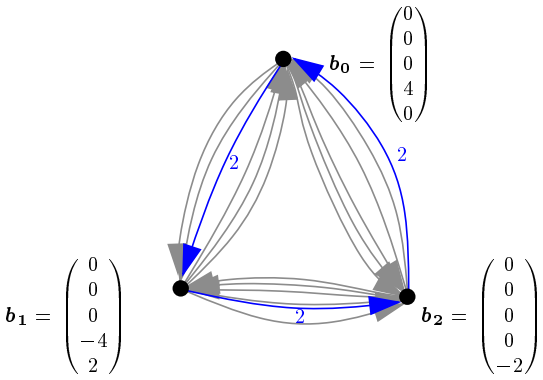
We call $A = V \times V \times L$ the set of *(directed) arcs*. With each directed arc $(ij\ell) \in A$ we associate a variable $x_{ij\ell} \in \mathbb{R}$ to indicate the number of times we fly using loading configuration ℓ from site i to site j. We call a vector $\boldsymbol{x} \in \mathbb{R}^{|A|}$ a *transportation plan*. For any set $S \subseteq V$ we use the notation $\delta^{\text{in}}(S) = \{(ij\ell) \mid j \in S, i \in V \setminus S, (ij\ell) \in A\}$ to indicate the set of arcs entering S, and $\delta^{\text{out}}(S) = \{(ij\ell) \mid i \in S, j \in V \setminus S, (ij\ell) \in A\}$ to indicate the set of arcs leaving S. For singleton sets $\{i\}$ we abbreviate this to $\delta^{\text{in}}(i)$ and $\delta^{\text{out}}(i)$. For any transportation plan $\boldsymbol{x}$ and $A' \subseteq A$, let $\boldsymbol{x}(A')$ denote the total flow over all arcs in A', i.e. $\boldsymbol{x}(A') = \sum_{(ij\ell) \in A'} \chi^\ell x_{ij\ell}$. The k^{th} component of $\boldsymbol{x}(A')$ is the total flow of commodity k over the arcs in A' and is denoted by $\boldsymbol{x}_k(A')$. A transportation plan $\boldsymbol{x} \geq 0$ is *feasible* if all planes that enter site i also leave site i, and if the difference between incoming and outgoing objects of each commodity at site i equals the production at site i, i.e. if

$$\boldsymbol{x}(\delta^{\text{out}}(i)) - \boldsymbol{x}(\delta^{\text{in}}(i)) = \boldsymbol{b}_i \qquad \forall i \in V. \tag{1}$$

Referring to the instance in Figure 1, we note that the solution $x_{01\ell_0} = x_{12\ell_1} = x_{20\ell_2} = 2$ is feasible. The *excess* of a transportation plan $\boldsymbol{x}$ is defined as the matrix $E(\boldsymbol{x}) = [e_{ik}(\boldsymbol{x}) | i \in V, k \in K]$, where row i of $E(\boldsymbol{x})$ (the *excess at site i given $\boldsymbol{x}$*) is given by $\boldsymbol{e}_i(\boldsymbol{x}) = \boldsymbol{b}_i + \boldsymbol{x}(\delta^{\text{in}}(i)) - \boldsymbol{x}(\delta^{\text{out}}(i))$.

We are given an asymmetric cost matrix $C \in \mathbb{N}^{n \times n}$, where c_{ij} denotes the cost of flying one loading configuration from i to j. In our case, c_{ij} is proportional to the time needed to fly a transportation aircraft from i to j. Define $\boldsymbol{c} \in \mathbb{N}^{|A|}$ to be the cost vector associated with the decision variables, where element $(ij\ell)$ of $\boldsymbol{c}$ is denoted $c_{ij\ell}$ and has value c_{ij}.

The Integer Airplane Problem (IPA) is to find a feasible transportation plan of minimum



$$V = \{0, 1, 2\}$$

$$K = \{k_0, \overbrace{k_1, k_2}^{\text{jigs}}, \overbrace{k_3, k_4}^{\text{parts}}\}$$

$$L = \{\ell_0, \ell_1, \ell_2\}, \quad \begin{aligned} \ell_0 &= \{k_0, \tfrac{k_3}{k_1}, \tfrac{k_3}{k_1}, k_2\}, \\ \ell_1 &= \{k_0, k_1, k_1, \tfrac{k_4}{k_2}\}, \\ \ell_2 &= \{k_0, k_1, k_1, k_2\} \end{aligned}$$

Figure 1: Example Instance of IPA.

cost:

$$\text{(IPA)} \quad \begin{array}{rll} \min & z(\boldsymbol{x}) = \boldsymbol{c}^T \boldsymbol{x} & \\ \text{subject to} & \boldsymbol{x}(\delta^{\text{out}}(i)) - \boldsymbol{x}(\delta^{\text{in}}(i)) = \boldsymbol{b}_i & \forall i \in V, \\ & \boldsymbol{x} \geq 0, \ \boldsymbol{x} \text{ integer.} & \end{array}$$

We denote the linear programming (LP) relaxation of IPA by LPA. The size of the problem formulation is not dominated by the number of sites or the number of commodities, which are relatively small, but by the size of L, which causes the number of $x_{ij\ell}$ variables to be very large, and the fact that the $x_{ij\ell}$ variables are general integer variables.

Problem IPA, as well as realistic data, were presented to us by ORTEC Consultants bv. The data set consists of parts from four different airplane types, and currently one basically solves a transportation problem for each airplane type. The primary goal of this research is to determine how much can be saved by combining the transport of the parts of the different airplane types. IPA differs from the operational problem that ORTEC solves for their client as IPA ignores all timing aspects. Problem LPA is of interest to ORTEC because it can be used to gain insight in the structure of low cost solutions.

Related literature. Problem IPA contains the NP-hard multicover problem [HH86] as follows. For each cut $(S, V \setminus S)$, at least the parts that are produced in S and not consumed in S have to be covered by sets associated with the admissible loading configurations going from S to $V \setminus S$. Furthermore, IPA is related to the NP-complete Integral Network Flow problem with Homologous Arcs [GJ79] as follows. Let $V := V \cup \{s, t\}$. For each $x_{ij\ell}$, make a set $H_{ij\ell}$ containing $\chi^\ell(k)$ arcs (i, j) for each commodity k. Let $E = \bigcup_{ij\ell} H_{ij\ell}$ and $\mathcal{H} = \bigcup_{ij\ell}\{H_{ij\ell}\}$. Furthermore connect the source node s to all nodes with production of some commodity, and all nodes that have demand of some commodity to the sink node t, using arcs that have capacity equal to the production/demand. Any solution to IPA gives an $s-t$ flow f in the graph (V, E) satisfying $f(a) = f(a')$ for each $a, a' \in H$, $H \in \mathcal{H}$. So, IPA can be seen as a problem that has multiple sets of homologous arcs, each set restricted to arcs that have the same source and destination.

The observations in the previous paragraph make it quite unlikely to find efficient polynomial time algorithms for IPA. Therefore, we propose to use branch and bound for solving IPA

to optimality. For this approach to work, we will need to be able to calculate good lower and upper bounds on the value of the solution in any node of the branch and bound tree. We will use LPA to find lower bounds, and an heuristic to find upper bounds. The branch and bound framework will impose lower- and upper bounds on the $x_{ij\ell}$ variables. With these lower- and upper bounds LPA can be seen as a generalisation of network flow [AMO93]. Network flow optimality conditions, and some network transformations do generalise to LPA, but we have been unable to generalise algorithms for network flow to LPA. Instead we apply LP column generation techniques. Although these techniques are well established they recently received a lot of attention, see e.g. Vanderbeck [Van94], Barnarth *et al.* [BJN+96], Sol [Sol94], and Desrochers *et al.* [DDS92]. The success of the column generation approach can be motivated by the improvement in hardware and in the quality of the available LP solvers, which allow much larger LP models to be solved. In order to get a fast and good primal approximation algorithm that can be used in each node of the branch and bound tree, we propose an LP-based heuristic, that converts a fractional LP solution to an integer solution for IPA. LP-based approximation algorithms have been quite successful in different problem domains. Examples of these are facility location problems [STA97] and scheduling with communication delays [MK97].

This paper is organised as follows. In Section 2 we consider a straightforward heuristic based on minimum cost flow. We discuss solving and "repairing" LPA in Section 3. Next, we describe how to combine these techniques in a branch and bound in order algorithm to compute provably optimal solutions in Section 4. We discuss some of the problems we encountered in trying to generalise network flow algorithms to LPA in Section 5. Experimental results are given in Section 6.

2 Constructing Solutions of IPA

In Section 2.1 we show how to construct solutions of IPA using a so-called *implementable order* by calculating a minimum cost flow for each commodity. An implementable order for our ORTEC data is given in Section 2.2. In Section 2.3 we consider the quality of the solutions returned by the construction introduced in Section 2.1.

2.1 Minimum Cost Flow

In this Section we present the Minimum Cost Flow (MCF) heuristic for IPA. We start by defining our main tool in constructing feasible solutions, namely *implementable orders*. The main difficulty in finding feasible solutions for IPA lies in the fact that we can not independently ship our commodities. An implementable order is an order in which to ship our commodities such that we can ship any commodity independently of all earlier commodities. More precisely, an implementable order consists of a precedence relation $\prec$ over K and a vector-valued function $\mathrm{lv}(k,q)$ with a component $\mathrm{lv}_\ell(k,q)$ for each $\ell \in L$, that tells us how to properly implement amount q of flow of commodity k using admissible loading configurations:

Definition 2.1 *A tuple* $(\prec, \mathrm{lv})$*, where* $\prec$ *is a linear order on* K *and* $\mathrm{lv} : K \times \mathbb{N} \to \mathbb{N}^{|L|}$*, is an* implementable order *if and only if for all commodities* $k \in K$*, and for all quantities* $q \in \mathbb{N}$ *the following holds:*

$$\sum_{\ell \in L} \mathrm{lv}_\ell(k,q)\chi^\ell(k) = q, \qquad \text{and} \tag{2a}$$

$$\sum_{\ell \in L} \mathrm{lv}_\ell(k,q)\chi^\ell(k') = 0, \qquad \forall k' \in K : k' \prec k. \tag{2b}$$

```
procedure IPAbyMinCostFlow(B ∈ ℤ^{n×d})
begin x = 0;
      Let (≺, lv) be an implementable order, and k_(1) ≺ k_(2) ≺ ··· ≺ k_(d);
      for k := k_(1) to k_(d) do
        begin Compute a minimum cost flow f for e_{·k}(x);
              Calculate x' = [x'_{ijℓ} | (ijℓ) ∈ A] with x'_{ijℓ} := lv_ℓ(k, f(i, j));
              Set x := x + x'
        end
      return x
end
```

Algorithm 2: The Minimum Cost Flow Heuristic.

Equation (2a) expresses that precisely q units of commodity k are transported. Equation (2b) enforces that this transportation is done independently of all commodities that precede k. Problem specific knowledge has to be used to provide such an order. We give an implementable order for our ORTEC datasets in Section 2.2.

By using an implementable order it is easy to construct a feasible solution $\boldsymbol{x}$. Let $k_{(1)}, \ldots, k_{(d)}$ be a permutation of K such that for $1 \leq i < j \leq d$ we have $k_{(i)} \prec k_{(j)}$. We construct a feasible solution in d stages, numbered from 1 to d. Initially, $\boldsymbol{x} = 0$. The goal of stage s is to transport all positive excess of commodity $k_{(s)}$ in $\boldsymbol{x}$ to the sites with negative excess of $k_{(s)}$ in $\boldsymbol{x}$. This is done as follows. First, a minimum cost flow f is computed for the production/demand vector $\boldsymbol{e}_{\cdot k_{(s)}}(\boldsymbol{x})$ on the complete graph on V with cost matrix C. This minimum cost flow is implemented in a transportation plan $\boldsymbol{x}' = [x'_{ij\ell} | (ij\ell) \in A]$ where $x'_{ij\ell} := \mathrm{lv}_\ell(k_{(s)}, f(i,j))$. Stage s ends by setting $\boldsymbol{x} := \boldsymbol{x} + \boldsymbol{x}'$. Pseudo code can be found in Algorithm 2. For the computation of the minimum cost flows, we use the cost scaling algorithm by Goldberg and Tarjan [GT90], which we implemented using LEDA graphs [MNU96].

Lemma 2.2 *The MCF heuristic returns a feasible solution of IPA.*

Proof. The lemma can be proved by induction on the stage number s, where stage 0 denotes the initialisation. Let $\boldsymbol{x}^s$ denote the value of $\boldsymbol{x}$ at the end of stage s. Our induction hypothesis is that at the end of stage s we have

$$e_{ik}(\boldsymbol{x}^s) = 0 \qquad \forall i \in V, k \in K : k \prec k_{(s)}.$$

For $s = 0$ the induction hypothesis is trivially true, so suppose that we have $s > 0$ and that the lemma holds for stage $s-1$. Let $\boldsymbol{x}'$ be the vector added to $\boldsymbol{x}^{s-1}$ in stage s. Focus on any site i. Pick $k \prec k_{(s)}$ arbitrarily. By (2b) we have $\boldsymbol{x}'_k(\delta^{\mathrm{in}}(i)) - \boldsymbol{x}'_k(\delta^{\mathrm{out}}(i)) = 0$. By induction we have $e_{ik}(\boldsymbol{x}^{s-1}) = 0$. Hence,

$$\begin{aligned} e_{ik}(\boldsymbol{x}^s) &= b_{ik} + \boldsymbol{x}^s_k(\delta^{\mathrm{in}}(i)) - \boldsymbol{x}^s_k(\delta^{\mathrm{out}}(i)) \\ &= b_{ik} + \boldsymbol{x}^{s-1}_k(\delta^{\mathrm{in}}(i)) - \boldsymbol{x}^{s-1}_k(\delta^{\mathrm{out}}(i)) + \boldsymbol{x}'_k(\delta^{\mathrm{in}}(i)) - \boldsymbol{x}'_k(\delta^{\mathrm{out}}(i)) \\ &= e_{ik}(\boldsymbol{x}^{s-1}) + 0 = 0. \end{aligned}$$

As this holds for all possible choices of $k \in K : k \prec k_{(s)}$, it remains to prove the induction for commodity $k_{(s)}$.

Let f be the minimum cost flow computed in stage s, and let $k = k_{(s)}$. By feasibility of f, we have that $\sum_{j\in V} f(i,j) - \sum_{j\in V} f(j,i) = e_{ik}(\boldsymbol{x}^{s-1})$. By (2a) we have that $\boldsymbol{x}'_k(A(j,i)) = f(j,i)$ and that $\boldsymbol{x}'_k(A(i,j)) = f(i,j)$. Now observe that for all sites i

$$\begin{aligned} e_{ik}(\boldsymbol{x}^s) &= b_{ik} + \boldsymbol{x}_k^{s-1}(\delta^{\text{in}}(i)) - \boldsymbol{x}_k^{s-1}(\delta^{\text{out}}(i)) + \boldsymbol{x}'_k(\delta^{\text{in}}(i)) - \boldsymbol{x}'_k(\delta^{\text{out}}(i)) \\ &= e_{ik}(\boldsymbol{x}^{s-1}) + \sum_{j\in V} f(j,i) - \sum_{j\in V} f(i,j) \\ &= e_{ik}(\boldsymbol{x}^{s-1}) - e_{ik}(\boldsymbol{x}^{s-1}) = 0. \end{aligned}$$

This proofs that the solution satisfies the flow conservation constraints (1). Furthermore, as the returned $\boldsymbol{x}$ is a sum of positive integer vectors, it also satisfies $\boldsymbol{x} \geq 0$ and integrality. This completes the proof of the lemma. □

Lemma 2.3 *The MCF heuristic can be implemented to work in* $\mathrm{O}(dn^3 \log \bar{C})$ *time, where* $\bar{C} = \max_{i,j\in V} c_{ij}$.

Proof. The complexity follows from the fact that the heuristic computes d minimum cost flows, each of which can be computed in $\mathrm{O}(n^3 \log \bar{C})$ time [GT90]. □

2.2 An Implementable Order

Let $c_k = \max_{\ell\in L} \chi^\ell(k)$ be the maximal number of times an element of commodity k occurs in any admissible loading configuration. The *preferred loading configuration* for transporting q units of commodity k, denoted $\text{pref}_k(q)$, is defined as follows:

$$\text{pref}_k(q) = \begin{cases} \{k_0\} \cup \bigcup_1^{\min(q,c_k)} \{\text{trailer}(k), k\} & \text{if } k \text{ is a part,} \\ \{k_0\} \cup \bigcup_1^{\min(q,c_k)} \{k\}, & \text{if } k \text{ is a trailer, and} \\ \{k_0\}, & \text{if } k = k_0. \end{cases}$$

The technical assumption referred to in Section 1 is that for our ORTEC datasets $\text{pref}_k(q)$ is admissible for all $k \in K, q \in \mathbb{N}$. Using these preferred loading configurations, our implementable order $(\prec, \text{lv})$ is given by $k_{(1)} \prec \cdots \prec k_{(j-1)} \prec k_{(j)} \prec \cdots \prec k_{(d-1)} \prec k_{(d)} = k_0$, where $\{k_{(1)}, \ldots, k_{(j-1)}\}$ is the set of all parts, and $\{k_{(j)}, \ldots, k_{(d-1)}\}$ the set of all trailers. Furthermore, $\text{lv} : K \times \mathbb{N} \to \mathbb{N}^{|L|}$ is given by

$$\forall \ell \in L : \text{lv}_\ell(k,q) = \begin{cases} \lfloor q/c_k \rfloor & \text{if } \ell = \text{pref}_k(q) \\ 1 & \text{if } \ell = \text{pref}_k(q \bmod c_k) \wedge q \bmod c_k > 0 \\ 0 & \text{otherwise} \end{cases}$$

Now we have $\sum_{\ell\in L} \text{lv}_\ell(k,q)\chi^\ell(k) = \lfloor q/c_k \rfloor \cdot c_k + q \bmod c_k = q$. Because a preferred loading configuration for commodity k does not contain items of commodity k' for any $k, k' \in K : k' \prec k$, we also have $\sum_{\ell\in L} \text{lv}_\ell(k,q)\chi^\ell(k') = 0$. Hence, $(\prec, \text{lv})$ as above is indeed an implementable order. This allows us to efficiently construct feasible solutions for our ORTEC data sets.

2.3 Quality of the Minimum Cost Flow Solution

Let $|\ell| = \sum_{k \in K} \chi^\ell(k)$ denote the size of ℓ and let $\ell^{\max} = \max_{\ell \in L} |\ell|$ denote the maximum size of any loading configuration. In this section we prove that the MCF heuristic is an $O(\ell^{\max})$-approximation algorithm when used with the order of Section 2.2, assuming that $c_{ij} = O(c_{ji})$.

We start by introducing some notation. Let $\tilde{\boldsymbol{x}}$ denote a feasible solution returned by the MCF heuristic, and $\boldsymbol{x}^*$ an optimal solution to IPA. Define $\text{cost}_k(\boldsymbol{x})$, the cost assigned to commodity k in $\boldsymbol{x}$, as

$$\text{cost}_k(\boldsymbol{x}) = \sum_{(ij\ell) \in A} c_{ij} \chi^\ell(k) x_{ij\ell} / |\ell|.$$

Using this expression we have $\sum_{k \in K} \text{cost}_k(\boldsymbol{x}) = z(\boldsymbol{x})$. For part k (and $k = k_0$), let f_k^* denote a minimum cost flow for the production/demand vector $\boldsymbol{b}_{\cdot k}$. For trailer k, let $f_k^* = f_k' + \sum_{k' \in \text{parts}(k)} f_{k'}^*$, where f_k' is a minimum cost flow for the production/demand vector $\boldsymbol{b}_{\cdot k} - \sum_{k' \in \text{parts}(k)} \boldsymbol{b}_{\cdot k'}$. For $k \in K$, let v_k^* denote the value of f_k^*. Define $A(i,j)$ as the set of arcs going from i to j, i.e. $A(i,j) = \{(ij\ell) \,|\, (ij\ell) \in A\}$. Let $f_k^{\boldsymbol{x}}$ denote the flow of commodity k in $\boldsymbol{x}$, the components of $f_k^{\boldsymbol{x}}$ are $f_k^{\boldsymbol{x}}(i,j) = \boldsymbol{x}_k(A(i,j))$. We will first show that $z(\boldsymbol{x}^*) \geq 1/\ell^{\max} \cdot \sum_{k \in K} v_k^*$. Next we will show that $z(\tilde{\boldsymbol{x}}) = O(\sum_{k \in K} v_k^*)$ assuming $c_{ij} = O(c_{ji})$. Together this gives the desired result.

Lemma 2.4 $z(\boldsymbol{x}^*) \geq \dfrac{1}{\ell^{\max}} \sum_{k \in K} v_k^*$.

Proof. Observe that for any k, $f_k^{\boldsymbol{x}^*}$ satisfies the same constraints as f_k^*, namely, the flow conservation constraints for all commodities together with the implied flow for trailers. Moreover, since the f_k^* are the minimum cost structures satisfying these constraints, the cost of $f_k^{\boldsymbol{x}^*}$ is at least v_k^*. These observations imply that

$$\text{cost}_k(\boldsymbol{x}^*) = \sum_{(ij\ell) \in A} \frac{c_{ij} \chi^\ell(k) x_{ij\ell}^*}{|\ell|} \geq \sum_{i,j \in V} \frac{c_{ij} \boldsymbol{x}_k^*(A(i,j))}{\ell^{\max}} = \frac{1}{\ell^{\max}} \sum_{i,j \in V} c_{ij} f_k^{\boldsymbol{x}^*}(i,j) \geq v_k^* / \ell^{\max}.$$

We can now conclude that

$$z(\boldsymbol{x}^*) = \sum_{k \in K} \text{cost}_k(\boldsymbol{x}^*) \geq \frac{1}{\ell^{\max}} \sum_{k \in K} v_k^*.$$

□

Lemma 2.5 $z(\tilde{\boldsymbol{x}}) = O(\sum_{k \in K} v_k^*)$.

Proof. Observe that for parts k (trailers k) $f_k^{\tilde{\boldsymbol{x}}}$ is constructed as (a combination of) minimum cost flows. This implies that the cost of $f_k^{\tilde{\boldsymbol{x}}}$, $\sum_{i,j \in V} c_{ij} f_k^{\tilde{\boldsymbol{x}}}(i,j)$, equals v_k^*. Therefore, we have that for all $k \in K, k \neq k_0$ that

$$\text{cost}_k(\tilde{\boldsymbol{x}}) = \sum_{(ij\ell) \in A} \frac{c_{ij} \chi^\ell(k) \tilde{x}_{ij\ell}}{|\ell|} \leq \sum_{i,j \in V} c_{ij} \tilde{\boldsymbol{x}}_k(A(i,j)) = \sum_{i,j \in V} c_{ij} f_k^{\tilde{\boldsymbol{x}}}(i,j) = v_k^*.$$

It remains to bound the value of $\text{cost}_{k_0}(\tilde{\boldsymbol{x}})$. Recall from Lemma 2.2 that, by construction, $\tilde{\boldsymbol{x}} = \boldsymbol{x}^{d-1} + \boldsymbol{x}'$. Hence $\text{cost}_{k_0}(\tilde{\boldsymbol{x}}) = \text{cost}_{k_0}(\boldsymbol{x}^{d-1}) + \text{cost}_{k_0}(\boldsymbol{x}')$. Clearly $\chi^\ell(k_0) = 1 \leq \chi^\ell(k)$ for any $k \in \ell$ implies that

$$\text{cost}_{k_0}(\boldsymbol{x}^{d-1}) \leq \sum_{k \in K, k \neq k_0} \text{cost}_k(\boldsymbol{x}^{d-1}) = \sum_{k \in K, k \neq k_0} v_k^*.$$

Now observe that, given the transportation plan $\boldsymbol{x}^{d-1}$, a feasible solution for IPA can be constructed by adding one empty transportation plane $x_{ji\{k_0\}}$ for each unit $x_{ij\ell}$, together with a minimum cost flow of empty transportation planes for the production/demand vector $\boldsymbol{b}_{\cdot k_0}$. The cost of this feasible solution is an upper bound on the cost of $\boldsymbol{x}'$. By our assumption that $c_{ij} = \text{O}(c_{ji})$ we obtain:

$$\text{cost}_{k_0}(\boldsymbol{x}') \leq \sum_{k \neq k_0} \sum_{i,j \in V} c_{ji} f_k^{\tilde{\boldsymbol{x}}}(i,j) + v_{k_0}^* = \text{O}(\sum_{k \in K, k \neq k_0} v_k^*) + v_{k_0}^* = \text{O}(\sum_{k \in K} v_k^*).$$

Hence,

$$z(\tilde{\boldsymbol{x}}) = \sum_{k \in K} \text{cost}_k(\tilde{\boldsymbol{x}}) = \text{O}(\sum_{k \in K, k \neq k_0} v_k^*) + \text{O}(\sum_{k \in K} v_k^*) = \text{O}(\sum_{k \in K} v_k^*),$$

which concludes the proof. □

Theorem 2.6 *Assuming that $c_{ij} = \text{O}(c_{ji})$, the MCF heuristic is an $\text{O}(\ell^{\max})$-approximation algorithm for IPA that runs in $\text{O}(dn^3 \log \bar{C})$ time, where $\bar{C} = \max_{i,j} c_{ij}$.*

Proof. From Lemma 2.2 and Lemma 2.3 we know that the solution obtained by the MCF heuristic is feasible and that the running time is according to the claim. By Lemma 2.4 and lemma 2.5 we have that $z(\tilde{\boldsymbol{x}})/z(\boldsymbol{x}^*) = \text{O}(\ell^{\max})$. Together this proves the theorem. □

3 The LP Relaxation of IPA

In Section 3.1 we show how to solve LPA by column generation, which yields a fractional solution. We present a data structure to speed up the column generation in Section 3.2. In Section 3.3 we discuss alternative optimality conditions for LPA and derive a more sophisticated column generation scheme based on these optimality conditions. To construct an integer solution we first round the fractional solution and then use the MCF heuristic to restore feasibility. This *round and repair* approach is described in Section 3.4. The quality of the resulting integer solutions is significantly better than the quality of the solutions obtained after applying the MCF heuristic only. In this section, we impose without loss of generality an upper bound $\boldsymbol{u}$ on the $\boldsymbol{x}$-variables.

3.1 Solution by LP Column Generation

The basic column generation approach is described in [PS82, Chapter 4]. The idea behind LP column generation is the following. Instead of using all decision variables, we use only a small subset and calculate an optimal LP solution for this restricted formulation. The initial subset should be chosen in such a way that the restricted problem is feasible (assuming the original

procedure LPAbyLPColumnGeneration($B \in \mathbb{Z}^{n \times d}$)
begin Let $\boldsymbol{x}$ be an initial solution, $A = \{(ij\ell) : x_{ij\ell} > 0\}$;
  repeat Solve the LP relaxation (3), giving $\boldsymbol{\pi}$ and $\boldsymbol{x}$
    forall $i, j \in V$, $k \in K : i \neq j$ **do**
      begin let $\ell^* = \arg\min_{\ell \in L : k \in \ell, (ij\ell) \notin A} c^{\boldsymbol{\pi}}_{ij\ell}$;
        if $c^{\boldsymbol{\pi}}_{ij\ell^*} < 0$ **then** add $(ij\ell^*)$ to A
      end
  until no edges added to A;
  return $\boldsymbol{x}$
end

Algorithm 3: LP Relaxation of IPA by Column Generation.

problem is feasible). As the convex hull of feasible solutions to the restricted problem is fully contained in that of the complete problem, the optimal solution to the restricted problem does not give us an optimal solution to the complete problem in general. Therefore, LP optimality conditions are checked for all the variables that were left out of the restricted problem, which involves computing the reduced cost of these variables using an optimal solution to the dual of the restricted problem. The variables having negative reduced cost violate the LP optimality conditions. To avoid increasing the size of the formulation too much we add only a small subset of them to the restricted problem. The process is repeated until all variables satisfy the LP optimality conditions.

To represent a restricted set of variables, we define the notion of a *loading configuration graph* $D = (V, L, A)$, where V is a set representing the nodes of the graph (which in our case are the sites of the problem), L a set of loading configurations, and A a set of arcs from $V \times V \times L$. We use $\delta^{\text{in}}_D(S) = \delta^{\text{in}}(S) \cap A$, and $\delta^{\text{out}}_D(S) = \delta^{\text{out}}(S) \cap A$ to denote the arcs entering and leaving set $S \subseteq V$. The problem formulation restricted to loading configuration graph D is the following:

$$\begin{array}{rl} \min & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & \boldsymbol{x}(\delta^{\text{out}}_D(i)) - \boldsymbol{x}(\delta^{\text{in}}_D(i)) = \boldsymbol{b}_i \quad \forall i \in V \\ & 0 \leq \boldsymbol{x} \leq \boldsymbol{u}. \end{array} \tag{3}$$

The complete problem LPA corresponds to taking $A = V \times V \times L$.

Given a primal feasible solution $\boldsymbol{x}$ to LPA, we choose our initial arc set as $A = \{(ij\ell) : x_{ij\ell} > 0\}$. Solving the restricted LP relaxation (3) yields a dual variable $\boldsymbol{\pi}$ associated with the flow conservation constraints. Define the (LP) reduced cost $c^{\boldsymbol{\pi}}_{ij\ell}$ of arc $(ij\ell)$ as $c_{ij} - (\boldsymbol{\pi}_i - \boldsymbol{\pi}_j)^T \chi^\ell$. We use the following heuristic to generate edges: for each triple (i, j, k) with $i, j \in V$, $i \neq j$, $k \in K$ add an arc $(ij\ell) \notin A$ with $k \in \ell$ to A that minimises $c^{\boldsymbol{\pi}}_{ij\ell}$. See Algorithm 3.

The proposed column generation scheme does not guarantee that the value of the solution to the restricted problem (3) decreases in each iteration. A column generation scheme that achieves primal progress in each iteration without adding too much variables would be preferable over the proposed one. Such a column generation scheme would involve finding augmenting subgraphs (i.e., satisfying flow conservation and non-negativity constraints), similar to finding augmenting paths for network flow problems. Difficulties that we encountered

in devising such a scheme are reported in Section 5.

3.2 The Pricing Problem Revisited: kd-Trees

In this section we present a data structure that can be used to find the set of variables to be added to the restricted formulation more efficiently in practice. Recall that, for each commodity $k \in K$, we want to find a loading configuration ℓ that contains k and minimises $c^{\boldsymbol{\pi}}_{ij\ell}$ (we call this the *pricing problem*). Let $\boldsymbol{\pi}(i,j) = \boldsymbol{\pi}_j - \boldsymbol{\pi}_i$. Because c_{ij} is a constant, the pricing problem for commodity k from i to j can be formulated as follows:

$$\operatorname*{arg\,min}_{\ell \in L : \ell \ni k, (ij\ell) \notin A} \boldsymbol{\pi}(i,j)^T \chi^\ell. \tag{4}$$

In other words, we want to find loading configurations with characteristic vectors that are minimal with respect to the vector $\boldsymbol{\pi}(i,j)$. The data structure allows a hierarchical way of computing the prices of the individual loading configurations, and an ordering of all loading configurations that allow for sharing of computational effort and pruning the set of configurations for which the complete price has to be computed.

The data structure is an adapted version of the kd-tree [BKOS97]. Each node of the tree corresponds to a set of loading configurations. The set associated with node p is denoted L_p. In the root of the tree, we store the bounding box of all characteristic vectors. This involves storing two d-dimensional vectors $\boldsymbol{x}^{\min}(L)$ and $\boldsymbol{x}^{\max}(L)$, where

$$x^{\min}_k(L) = \min_{\ell \in L} \chi^\ell(k) \qquad\qquad x^{\max}_k(L) = \max_{\ell \in L} \chi^\ell(k).$$

An internal node has either two or three children. The sets of loading configurations associated with the children of an internal node p partition L_p. For some dimension d_p in the affine hull of the characteristic vectors of L_p, let b_p be the median of the multi set $\{\chi^\ell(d_p) \mid \ell \in L_p\}$. We associate the set of loading configurations $L_{\text{left}} = \{\ell \in L_p \mid \chi^\ell(d_p) < b_p\}$ with the *left* child of p, the set of loading configurations $L_{\text{middle}} = \{\ell \in L_p \mid \chi^\ell(d_p) = b_p\}$ with the *middle* child of p, and the set of loading configurations $L_{\text{right}} = \{\ell \in L_p \mid \chi^\ell(d_p) > b_p\}$ with the *right* child of p. In node p we store d_p, and pointers to the left, middle and right children of node p together with tuples $(\boldsymbol{\delta}^{\min}_{\text{left}}, \boldsymbol{\delta}^{\max}_{\text{left}})$, $(\boldsymbol{\delta}^{\min}_{\text{middle}}, \boldsymbol{\delta}^{\max}_{\text{middle}})$, and $(\boldsymbol{\delta}^{\min}_{\text{right}}, \boldsymbol{\delta}^{\max}_{\text{right}})$, where

$$\boldsymbol{\delta}^{\min}_{\text{left}} = \boldsymbol{x}^{\min}(L_{\text{left}}) - \boldsymbol{x}^{\min}(L_p), \qquad\qquad \boldsymbol{\delta}^{\max}_{\text{left}} = \boldsymbol{x}^{\max}(L_p) - \boldsymbol{x}^{\max}(L_{\text{left}}),$$

and where $\boldsymbol{\delta}^{\min}_{\text{middle}}, \boldsymbol{\delta}^{\max}_{\text{middle}}, \boldsymbol{\delta}^{\min}_{\text{right}}, \boldsymbol{\delta}^{\max}_{\text{right}}$ are defined similarly for the middle and right child of node p. Using these definitions, the bounding box of a child of node p can be derived from the bounding box of node p by adding the appropriate $\boldsymbol{\delta}^{\min}$ to $\boldsymbol{x}^{\min}(L_p)$ and subtracting the appropriate $\boldsymbol{\delta}^{\max}$ from $\boldsymbol{x}^{\max}(L_p)$. A leaf node of the tree corresponds to a singleton set of loading configurations and we only store this loading configuration. Finally, we choose the values of d_p in such a way that on any path from the root of the tree to a leaf of the tree, the d_p values associated with the nodes on the path iterate over the dimensions of the affine hull of characteristic vectors of the corresponding L_p. Pseudo code for building the tree is given in Algorithm 4.

This data structure can be used for solving the pricing problem (4) from i to j as follows. Let $\boldsymbol{\pi}^+(i,j)$ (and $\boldsymbol{\pi}^-(i,j)$) be the vector with components $\pi_k(i,j)$ if $\pi_k(i,j) > 0$ (and $\pi_k(i,j) < 0$, respectively) and 0 otherwise. The price of the cheapest corner of the bounding box in any node p is a lower bound for the price of all the loading configurations in the subtree

procedure BuildTree(L)
begin if $L = \{\ell\}$ **then** make a leaf node with corresponding loading configuration ℓ.
else begin
Calculate $\boldsymbol{x}^{\min}(L)$ and $\boldsymbol{x}^{\max}(L)$;
Choose d_p with $x^{\min}_{d_p}(L) < x^{\max}_{d_p}(L)$, calculate b_p;
Recursively build the subtrees on L_{left}, L_{middle}, and L_{right};
Calculate and store $\boldsymbol{\delta}^{\min}_{\text{left}}, \boldsymbol{\delta}^{\max}_{\text{left}}, \boldsymbol{\delta}^{\min}_{\text{middle}}, \boldsymbol{\delta}^{\max}_{\text{middle}}, \boldsymbol{\delta}^{\min}_{\text{right}}, \boldsymbol{\delta}^{\max}_{\text{right}}$
end
end

Algorithm 4: Building a kd-Tree on the Loading Configurations.

of p. We recursively visit the nodes of the tree starting in the root, and keep track of the price of the cheapest corner of the bounding box of the current node p. This price equals

$$q = \boldsymbol{\pi}^+(i,j)^T \boldsymbol{x}^{\min}(L_p) + \boldsymbol{\pi}^-(i,j)^T \boldsymbol{x}^{\max}(L_p),$$

and can be calculated easily if p is the root node of the tree. If q is larger than the best price for all $k \in K$ of all the loading configurations associated with the leaf nodes that were visited before visiting the current node, we are done in the current node. If not, we still might find an improvement for some k in the subtree of node p. Therefore, we recursively visit the children of node p. If $d_p > 0$, we first visit the left child of node p, then the middle one, and finally the right one. If $d_p \leq 0$, we first visit the right child of node p, then the middle one, and finally the left one. The price of the cheapest corner of the bounding box of the left child of node p is given by

$$q_{\text{left}} = q + \boldsymbol{\pi}^+(i,j)^T \boldsymbol{\delta}^{\min}_{\text{left}} - \boldsymbol{\pi}^-(i,j)^T \boldsymbol{\delta}^{\max}_{\text{left}},$$

and similar functions give the price of the cheapest corner of the bounding box of the middle and right children of node p. The query procedure on the data structure is illustrated in Algorithm 5.

The query time of the data structure can be guaranteed to be $O(d|L|)$, which is not worse than enumerating all loading configurations. Actually, experimental evaluation suggests a reduction of the number of loading configurations inspected and of the number of multiplications needed to solve the pricing problem by a factor of approximately 0.1. Moreover, in a careful implementation the overhead of building and traversing the search tree is small. Finally, the memory requirements are of the same order as the memory needed for storing the set of loading configurations.

3.3 Refined Column Generation

Observe that the constraints of LPA that are induced by a single commodity together form a minimum cost flow problem for this commodity, and that minimum cost flow problems are easy problems in the sense that we can solve them efficiently using polynomial time algorithms. The column generation scheme proposed in the previous subsection does not exploit this structure. In this section we show how to decompose LPA into separate minimum cost flow

var $z \in \mathbb{R}^d$ ($*$ z_k is the price of the cheapest $\ell \ni k$ seen so far. $*$)

procedure query($\boldsymbol{\pi} \in \mathbb{R}^d, p \in T, q \in \mathbb{R}$)
begin if $\forall k : q \geq z_k$ **or** p is leaf node and $(ij\ell_p) \in A$ **then return** ;
if p is a leaf node **then begin** $\forall k \in \ell_p : z_k := \min(z_k, q)$; **return end**
if $\pi_{d_p} > 0$ **then**
begin query($\boldsymbol{\pi}$, left$_p$, $q + (\boldsymbol{\pi}^+)^T \boldsymbol{\delta}^{\min}_{\text{left}} - (\boldsymbol{\pi}^-)^T \boldsymbol{\delta}^{\max}_{\text{left}}$);
query($\boldsymbol{\pi}$, middle$_p$, $q + (\boldsymbol{\pi}^+)^T \boldsymbol{\delta}^{\min}_{\text{middle}} - (\boldsymbol{\pi}^-)^T \boldsymbol{\delta}^{\max}_{\text{middle}}$);
query($\boldsymbol{\pi}$, right$_p$, $q + (\boldsymbol{\pi}^+)^T \boldsymbol{\delta}^{\min}_{\text{right}} - (\boldsymbol{\pi}^-)^T \boldsymbol{\delta}^{\max}_{\text{right}}$)
end else begin
query($\boldsymbol{\pi}$, right$_p$, $q + (\boldsymbol{\pi}^+)^T \boldsymbol{\delta}^{\min}_{\text{right}} - (\boldsymbol{\pi}^-)^T \boldsymbol{\delta}^{\max}_{\text{right}}$)
query($\boldsymbol{\pi}$, middle$_p$, $q + (\boldsymbol{\pi}^+)^T \boldsymbol{\delta}^{\min}_{\text{middle}} - (\boldsymbol{\pi}^-)^T \boldsymbol{\delta}^{\max}_{\text{middle}}$);
query($\boldsymbol{\pi}$, left$_p$, $q + (\boldsymbol{\pi}^+)^T \boldsymbol{\delta}^{\min}_{\text{left}} - (\boldsymbol{\pi}^-)^T \boldsymbol{\delta}^{\max}_{\text{left}}$);
end
end

procedure pricing($T, \boldsymbol{\pi} \in \mathbb{R}^d$)
begin Let p be the root of T and q the price of the cheapest corner of the bounding box;
forall $k \in K$ **do** $z_k := -c_{ij}$; ($*$ We only want variables with $c^{\boldsymbol{\pi}}_{ij\ell} < 0$. $*$)
query($\boldsymbol{\pi}, p, q$)
end

Algorithm 5: Solving the Pricing Problem from i to j.

problems with cost vectors that are derived from the LP reduced cost of the variables. We proof that if we can not find negative cost circulations for these separate minimum cost flow problems, then we have an optimal solution to LPA. If we find negative cost circulations in the separate minimum cost flow problems, we also have found a set of variables of LPA with negative LP reduced cost that can be added to the problem formulation. We call this column generation strategy our *refined* column generation strategy. When compared to the column generation scheme proposed in the previous subsection, the refined strategy results in smaller LP formulations.

Suppose we are given a loading configuration graph D, a vector $\boldsymbol{x} \in \mathbb{R}^{|A|}$, and a dual vector $\boldsymbol{\pi} \in \mathbb{R}^{n \times d}$. We say that a transportation plan $\boldsymbol{x}'$ is *feasible with respect to* $\boldsymbol{x}$ if it satisfies its production/demand constraints, and if $-\boldsymbol{x} \leq \boldsymbol{x}' \leq \boldsymbol{u} - \boldsymbol{x}$. The *reduced value* of a transportation plan $\boldsymbol{x}'$ in D is given by $z^{\boldsymbol{\pi}}(\boldsymbol{x}') = \sum_{a \in A} c^{\boldsymbol{\pi}}_a x'_a$. We say a transportation plan $\boldsymbol{x}'$ is a *circulation* if for all $i \in V$ we have $\delta^{\text{in}}(i) = \delta^{\text{out}}(i)$.

Lemma 3.1 *Let transportation plan $\boldsymbol{x}$ be a feasible solution to (3). Then $\boldsymbol{x}$ is an optimal transportation plan if and only if $z^{\boldsymbol{\pi}}(\boldsymbol{x}') \geq 0$ for all circulations $\boldsymbol{x}'$ that are feasible with respect to $\boldsymbol{x}$.*

Proof. Suppose $\boldsymbol{x}$ is optimal, and let $\boldsymbol{x}'$ be a circulation in D that is feasible with respect to $\boldsymbol{x}$. Observe that $\boldsymbol{x} + \boldsymbol{x}'$ is a feasible solution to (3). But then

$$z(\boldsymbol{x}') = z(\boldsymbol{x} + \boldsymbol{x}' - \boldsymbol{x}) = z(\boldsymbol{x} + \boldsymbol{x}') - z(\boldsymbol{x}) \geq 0.$$

Because $\boldsymbol{x}'$ is a circulation, $\boldsymbol{x}'(\delta^{\text{in}}(i)) - \boldsymbol{x}'(\delta^{\text{out}}(i)) = 0$, so

$$\begin{aligned} z(\boldsymbol{x}') &= \sum_{(ij\ell)\in A} c_{ij\ell}x'_{ij\ell} - \sum_{i\in V} \boldsymbol{\pi}_i^T \boldsymbol{x}'(\delta^{\text{out}}(i)) + \sum_{i\in V} \boldsymbol{\pi}_i^T \boldsymbol{x}'(\delta^{\text{in}}(i)) \\ &= \sum_{(ij\ell)\in A} (c_{ij\ell} - \boldsymbol{\pi}_i^T\chi^\ell + \boldsymbol{\pi}_j^T\chi^\ell)x'_{ij\ell} \quad = \quad \sum_{(ij\ell)\in A} c^{\boldsymbol{\pi}}_{ij\ell}x'_{ij\ell} \quad = \quad z^{\boldsymbol{\pi}}(\boldsymbol{x}'). \end{aligned}$$

It follows that $z^{\boldsymbol{\pi}}(\boldsymbol{x}') \geq 0$.

On the other hand, suppose $z^{\boldsymbol{\pi}}(\boldsymbol{x}') \geq 0$ for all circulations $\boldsymbol{x}'$ in D that are feasible with respect to $\boldsymbol{x}$. Let $\boldsymbol{x}''$ be any feasible solution to (3), and let $\boldsymbol{x}' = \boldsymbol{x}'' - \boldsymbol{x}$. Clearly, $-\boldsymbol{x} \leq \boldsymbol{x}' \leq \boldsymbol{u} - \boldsymbol{x}$. Because $\boldsymbol{x}$ and $\boldsymbol{x}''$ were both feasible solutions to (3) we also have $\boldsymbol{x}'(\delta^{\text{in}}(i)) - \boldsymbol{x}'(\delta^{\text{out}}(i)) = 0$. So $\boldsymbol{x}'$ is a circulation in D that is feasible with respect to $\boldsymbol{x}$. But then we have

$$z^{\boldsymbol{\pi}}(\boldsymbol{x}'') = z^{\boldsymbol{\pi}}(\boldsymbol{x} + \boldsymbol{x}') = z^{\boldsymbol{\pi}}(\boldsymbol{x}) + z^{\boldsymbol{\pi}}(\boldsymbol{x}') \geq z^{\boldsymbol{\pi}}(\boldsymbol{x}),$$

and $\boldsymbol{x}$ is optimal. □

Observe that any feasible solution to (3) satisfies the inequality

$$\boldsymbol{x}_k(\delta^{\text{out}}(i)) \geq \sum_{k'\in \text{parts}(k)} |b_{ik'}| \tag{5}$$

for any trailer type k. Define the *level* $\Phi(\boldsymbol{x}) \in \mathbb{R}^{n\times d}$ as $\Phi(\boldsymbol{x}) = [\phi_{ik}(\boldsymbol{x}) \,|\, i \in V, k \in K]$ where $\phi_{ik}(\boldsymbol{x}) = x_k(\delta^{\text{out}}(i)) - \sum_{k'\in\text{parts}(k)} |b_{ik'}|$ if k is a trailer, and $\phi_{ik}(\boldsymbol{x}) = \boldsymbol{x}_k(\delta^{\text{out}}(i))$ otherwise. Let $A' = \{(ij\ell) \,|\, i, j \in V, \ell \in L, x_{ij\ell} < u_{ij\ell}\}$. We now derive d separate directed graphs $D^k = (V, A^k)$ from D, one for each commodity $k \in K$, together with lower and upper bounds $\boldsymbol{l}^k$ and $\boldsymbol{u}^k$, and a flow $\boldsymbol{x}^k$ that is the projection of $\boldsymbol{x}$ on D^k. For each $\boldsymbol{u}_k(A(i,j)) - \boldsymbol{x}_k(A(i,j)) > 0$, include an arc $a = (ij)$ of cost $c^k(a) = \min_{a'\in A(i,j):x_{a'}<u_{a'}} c^{\boldsymbol{\pi}}_{a'}/|\ell|$ (we choose one a' that achieves the minimum and call this the *associated arc* of a), upper bound $u^k(a) = 1$ and lower bound $l^k(a) = 0$, and set $x^k(a) = 0$. For each $(ij\ell) \in A$ with $k \in \ell$ and $x_{ij\ell} > 0$, include an arc $a = (ij)$ of cost $c^k(a) = c^{\boldsymbol{\pi}}_{ij\ell}/|\ell|$ upper bound $u^k(a) = x_{ij\ell}$ and lower bound $l^k(a) = -x_{ij\ell}$, and set $x^k(a) = x_{ij\ell}$.

For any trailer type k, if $\boldsymbol{x}'$ is a circulation in D that is feasible with respect to $\boldsymbol{x}$, then (5) implies that

$$\boldsymbol{x}'_k(\{a \in \delta^{\text{in}}_D(i) \,|\, x'_a > 0\}) + \boldsymbol{x}'_k(\{a \in \delta^{\text{out}}_D(i) \,|\, x'_a < 0\}) \geq -\phi_{ik}. \tag{6}$$

Let $\text{cost}^{\boldsymbol{\pi}}_k(\boldsymbol{x}) = \sum_{(ij\ell)} c^{\boldsymbol{\pi}}_{ij\ell}\chi^\ell(k)x_{ij\ell}/|\ell|$.

Lemma 3.2 *For any feasible transportation plan $\boldsymbol{x}$, if there exists a circulation $\boldsymbol{x}'$ that is feasible with respect to $\boldsymbol{x}$ in D with $z^{\boldsymbol{\pi}}(\boldsymbol{x}') < 0$, then for some $k \in K$ there exists a circulation f^k that satisfies (6), that is feasible with respect to $\boldsymbol{x}^k$ in D^k, and has $\sum_{a\in A^k} c^k(a)f^k(a) < 0$.*

Proof. Let $\boldsymbol{x}'$ be any circulation in D that is feasible with respect to $\boldsymbol{x}$. Observe that $z^{\boldsymbol{\pi}}(\boldsymbol{x}') = \sum_{k\in K} \text{cost}^{\boldsymbol{\pi}}_k(\boldsymbol{x}')$. But then $z^{\boldsymbol{\pi}}(\boldsymbol{x}') < 0$ implies there exists $k \in K$ such that $\text{cost}^{\boldsymbol{\pi}}_k(\boldsymbol{x}') < 0$. Choose k such that $\text{cost}^{\boldsymbol{\pi}}_k(\boldsymbol{x}') < 0$.

For each $\boldsymbol{x}'_k(\{a \in A(i,j) \,|\, x'_a > 0\}) > 0$, let $a = (i,j) \in A^k$ be the (unique) arc with $l^k(a) = 0$ and $u^k(a) = 1$ and let $f^k(a) = \boldsymbol{x}'_k(\{a \in A(i,j) \,|\, x'_a > 0\})$. For each $\boldsymbol{x}'_{ij\ell} < 0$, choose

an unused arc $a = (i, j) \in A^k$ with cost $c^k(a) = c^{\boldsymbol{\pi}}_{ij\ell}$ and lower bound $l^k(a) < 0$, and set $f^k(a) = x'_{ij\ell}$. Choose $0 < \alpha \leq 1$ such that $-\boldsymbol{l}^k \leq \alpha f^k \leq \boldsymbol{u}^k$. Because $\boldsymbol{x}'$ satisfies (6) and this property is preserved by the construction of f^k, αf^k also satisfies (6). By construction αf^k is feasible in D^k with respect to $\boldsymbol{x}^k$. Moreover $\sum_{a \in A^k} c^k(a) f^k(a) \leq \text{cost}^{\boldsymbol{\pi}}_k(\boldsymbol{x}') < 0$, which implies $\sum_{a \in A^k} c^k(a) \alpha f^k(a) < 0$. This completes the proof. □

We are now able to describe our refined column generation strategy. For each part type k and for $k = k_0$ we calculate a minimum cost circulation f^k in D^k. For each trailer type k we calculate a minimum cost circulation f^k in D^k satisfying (6). For (i, j, k) with $f^k(i, j) > 0$ and $c^k(i, j) < 0$ we add its associated arc to A. Furthermore, we can limit the number of arcs that are added to the formulation as follows. In addition to the lower bounds (6) we can also add an upper bound of 1 unit of flow that passes a node. This way, only $O(dn)$ arcs are added to the problem formulation in each iteration of the column generation process. Lower and upper bounds on the flow that passes a node can be handled by an operation called *node splitting*, see also the book on network flows [AMO93].

We implemented and tested this refined column generation scheme. When compared to the column generation scheme proposed in Section 3.1, the number of variables is reduced by a factor of approximately 0.6 on our test instances. The number of LPs solved was about the same. Unfortunately, the observed running times almost doubled due to the extra work that is needed to maintain the graphs D^k and to calculate the minimum cost flows.

3.4 Rounding and Repairing the LP Solution

After computing the LP solution $\boldsymbol{x}$, it can be rounded (component-wise) to an integer solution $\boldsymbol{x}_1$. In general, the rounded solution will have a nonzero excess matrix $E(\boldsymbol{x}_1)$, and $\boldsymbol{x}_1$ will not be a feasible solution to IPA. Fortunately, this can be repaired easily as follows. Calculate $\boldsymbol{x}_2$ by executing the MCF heuristic with production and demand matrix $E(\boldsymbol{x}_1)$. Let $\boldsymbol{x}' = \boldsymbol{x}_1 + \boldsymbol{x}_2$. It can be shown, similarly to the last part of the proof of Lemma 2.2, that $\boldsymbol{x}'$ is feasible. We call this strategy the "round and repair" approach to solving IPA.

Candidate rounding strategies are rounding to the closest integer vector, rounding up, rounding down or rounding randomised (i.e., rounding each component up with probability proportional to the fractional part and down otherwise). A good rounding strategy should minimise the excess in the rounded solution. Here we use rounding to the closest integer vector. For the instances in our datasets, it turned out that the solutions obtained this way were good, as can be seen in the experimental results in Section 6.

4 Solving IPA by Branch and Bound

We implemented a branch and bound algorithm for IPA in the ABACUS framework provided by Thienel [Thi95]. To obtain upper bounds on the value of the optimal solution, we apply the round and repair algorithm from Section 3.4 to the solutions of the LP relaxations that are computed in each node of the branch and bound tree. Using standard branching on variables $x_{ij\ell}$ was sufficient for finding provably optimal solutions for the smaller datasets. To handle the larger instances, we implemented and tested the following branching strategy: first make sure that the total amount of commodity k leaving a set $S \subsetneq V$ is integral for each k, then make sure the total amount of commodity k shipped from i to j over all loading configurations is integral for all i, j, k, and only when this is achieved try to establish integrality of the

procedure findCut($k \in K, S, T$) (* Initially $S = \varnothing$ and $T = V$ *)
begin if $T = \varnothing$ **then return** S;
Pick node $i \in T$ arbitrarily;
S_1 := findCut($k, S, T \setminus \{i\}$);
S_2 := findCut($k, S \cup \{i\}, T \setminus \{i\}$);
if $\boldsymbol{x}_k(\delta^{\text{out}}(S_1))$ is more fractional than $\boldsymbol{x}_k(\delta^{\text{out}}(S_2))$
then return S_1;
else return S_2;
end

Algorithm 6: Finding a Most Fractional Cut for Commodity k.

individual $x_{ij\ell}$ variables. This approach exploits the embedded hierarchy of the problem, and results in an LP model that uses upper and lower bounds on sets of variables.

Let $A(i,j) = \{(ij\ell) \,|\, (ij\ell) \in A\}$. We find $S \subset V$ such that $\boldsymbol{x}_k(\delta^{\text{out}}(S))$ is most fractional over all $k \in K$ using the backtracking algorithm in Algorithm 6. If no fractional cut can be found, we check all $\boldsymbol{x}_k(A(i,j))$ for all i,j,k. If $\boldsymbol{x}_k(A(i,j))$ is integer for all i,j,k we take $A' = \{(ij\ell)\}$ for some fractional $x_{ij\ell}$ and $k = k_0$. We branch on the constraints $\boldsymbol{x}_k(A') \le \lfloor f \rfloor$ and $\boldsymbol{x}_k(A') \ge \lceil f \rceil$. We maintain the tuples (A', k) associated with lower (upper) bounds on $\boldsymbol{x}_k(A')$ in the set $\mathcal{L}$ (and $\mathcal{U}$, respectively). In each node of the branch and bound tree, an LP model of the following type has to be solved:

$$\begin{aligned}
\min \quad & \boldsymbol{c}^T \boldsymbol{x} \\
\text{subject to} \quad & \boldsymbol{x}(\delta_D^{\text{out}}(i)) - \boldsymbol{x}(\delta_D^{\text{in}}(i)) = \boldsymbol{b}_i && \forall i \in V \\
& \boldsymbol{x}_k(A') \ge l_{A'k} && \forall (A',k) \in \mathcal{L} && \text{(7a)} \\
& -\boldsymbol{x}_k(A') \ge -u_{A'k} && \forall (A',k) \in \mathcal{U} && \text{(7b)} \\
& \boldsymbol{x} \ge 0,
\end{aligned}$$

where $l_{A'k}$ and $u_{A'k}$ denote the lower and upper bounds on the value of $\boldsymbol{x}_k(A')$. Let

$$\sigma^+_{A'k} = \begin{cases} \text{dual to (7a)} & \text{if } (A',k) \in \mathcal{L}, \\ 0, & \text{otherwise,} \end{cases} \qquad \sigma^-_{A'k} = \begin{cases} \text{dual to (7b)} & \text{if } (A',k) \in \mathcal{U}, \\ 0, & \text{otherwise,} \end{cases}$$

and let $\boldsymbol{\sigma} = \boldsymbol{\sigma}^+ - \boldsymbol{\sigma}^-$. Denote the collection of arc sets A' associated either with branching on a cut ($A' = \delta^{\text{out}}(S)$ for some $S \subset V$) or with branching on the total flow of some commodity ($A' = A(i,j)$ for some $i,j \in V$) by $\mathcal{A}'$. The reduced cost used in the column generation process now becomes $c^{\boldsymbol{\pi},\boldsymbol{\sigma}}_{ij\ell} = c^{\boldsymbol{\pi}}_{ij\ell} - (\sum_{A' \in \mathcal{A}' : A'(i,j) \neq \varnothing} \boldsymbol{\sigma}_{A'})^T \chi^\ell$. The pricing problem in a node of the branch and bound tree can be reformulated as finding loading configurations with characteristic vectors that are minimal with respect to $\boldsymbol{\pi}(i,j) - \sum_{A' \in \mathcal{A}' : A'(i,j) \neq \varnothing} \boldsymbol{\sigma}_{A'}$. Hence, we can use the data structure of Section 3.2 to generate new variables. The branch and bound algorithm is summarised in Algorithm 7.

5 Problems in Generalising Network Flow Algorithms to LPA

Here, we explain the difficulties we encountered when trying to generalise network flow algorithms to LPA. We will first focus on the classical augmenting paths algorithm for network

procedure IPAbyBranch'nBound($B \in \mathbb{Z}^{n\times d}, \mathcal{L}, \mathcal{U}$)
begin Let $\tilde{\boldsymbol{x}}$ be the best integer primal seen so far;
Solve the LP relaxation (7) by column generation giving $\boldsymbol{x}$;
if LP feasible, $\boldsymbol{x}$ fractional, and $\boldsymbol{c}^T\boldsymbol{x} < \boldsymbol{c}^T\tilde{\boldsymbol{x}}$ **then**
begin Repair $\boldsymbol{x}$, check result against $\tilde{\boldsymbol{x}}$
if there exists $S \subset V, k \in K$ with fractional $\boldsymbol{x}_k(\delta^{\text{out}}(S))$
then Choose (A', k) such that $A' = \delta^{\text{out}}(S)$ and $\boldsymbol{x}_k(A')$ most fractional
else if there exists $i, j \in V, k \in K$ with fractional $\boldsymbol{x}_k(A(i,j))$
then Choose (A', k) such that $A' = A(i,j)$ and $\boldsymbol{x}_k(A')$ most fractional
else Choose most fractional $x_{ij\ell}$ and let $A' := \{(ij\ell)\}$ and $k := k_0$
Store $l_{A'k}$; $l_{A'k} := \lceil \boldsymbol{x}_k(A') \rceil$;
IPAbyBranch'nBound($\boldsymbol{b}, \mathcal{L} \cup \{(A', k)\}, \mathcal{U}$); restore $l_{A'k}$;
Store $u_{A'k}$; $u_{A'k} := \lfloor \boldsymbol{x}_k(A') \rfloor$;
IPAbyBranch'nBound($\boldsymbol{b}, \mathcal{L}, \mathcal{U} \cup \{(A', k)\}$); restore $u_{A'k}$
end
end

Algorithm 7: Branch and Bound Algorithm for IPA (by DFS).

flow, and then on preflow-push methods. Moreover, we concentrate on the problem of finding feasible flows for LPA in a loading configuration graph $D = (V, L, A)$ instead of minimum cost feasible flows. We assume the reader is familiar with the augmenting path algorithm by Ford and Fulkerson [FF62] and with the preflow-push algorithm by Goldberg and Tarjan [GT88].

Augmenting Paths. Let $e^+(\boldsymbol{x})$ denote the total positive excess in the current flow $\boldsymbol{x}$, i.e. $e^+(\boldsymbol{x}) = \frac{1}{2}\sum_{i\in V, k\in K} |e_{ik}(\boldsymbol{x})|$. Consider the class of problems LPA for which $d = 1$ on a loading configuration graph $D = (V, L, A)$. This class is equal to network flow on $D = (V, A)$ if we ignore the loading configurations. Any augmenting path algorithm for network flow maintains a flow $\boldsymbol{x}$. Each iteration a path from a source node to a sink node in the residual graph of $\boldsymbol{x}$ is found, and this path is added to $\boldsymbol{x}$. Now observe that the value of $e^+(\boldsymbol{x})$ strictly decreases over each iteration, a condition that eventually gives termination of the algorithm.

Such a construction is not possible for LPA if $d \geq 3$. For example, consider the problem instance in Figure 8. Clearly, it contains only one augmenting path, which goes from node 0 to node 1 using arc $(0, 1, \ell_0)$. Increasing the flow on this paths by $\epsilon > 0$ also increases $e^+(\boldsymbol{x})$ by ϵ. From this example we conclude that it is not possible to find a feasible flow by iteratively augmenting flow along a path from a source node to a sink node in such a way that the total positive excess strictly decreases.

Preflow-Push. One might argue that the above problem does not occur in the more sophisticated preflow-push methods, where termination is not obtained directly from the decrease in $e^+(\boldsymbol{x})$, but also from an upper bound on the maximal value of any node label. Informally, termination of this kind of algorithm is derived in two steps, the first step is the convergence of the node labels to some (node-specific) maximum value, the second is pushing flow from nodes with excess to nodes that have lower node labels, until all excess is zero. In the network flow case, it is possible to show that at any stage during the execution of the algorithm and for any node i with $e_i(\boldsymbol{x}) > 0$ there is a node j with $e_j(\boldsymbol{x}) < 0$ that is reachable from i

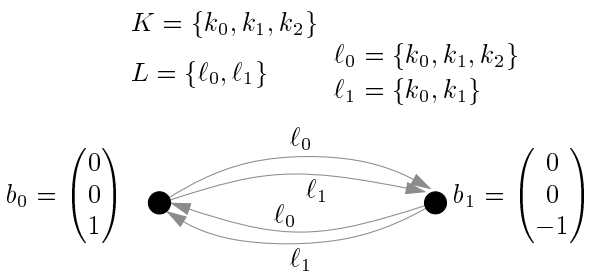


Figure 8: Counter Example.

in the residual graph of the current pseudo flow. Moreover node j has had $e_j(\boldsymbol{x}) < 0$ from the beginning of the algorithm. So j has never been relabelled during the execution of the algorithm and therefore still has its initial node label. For each arc (i, j) in the residual graph of the pseudo flow, the reduced cost of (i, j) (defined as $c_{ij} - \pi_i + \pi_j$, where π_i and π_j are the node labels of i and j) is bounded by the construction of the algorithm at any time during the execution. Consequently, the minimum path length to a node with negative excess in the residual graph of the current pseudo flow is a maximum on any node label.

Such a construction is difficult to realise in LPA because of the inability to ship commodities independently. To see this, consider the example in Figure 8. Any push in this graph introduces new negative excess. In principle, any node of the graph with a positive excess of some commodity at a given iteration of the algorithm can have negative excess of the same commodity at a later iteration, and vice versa. In the iterations that the excess of some commodity at a node is positive, the corresponding node label can be increased, so no node label qualifies as a node label that will not be increased during the execution. This in turn destroys the proof of termination.

6 Experimental Results

Experimental study of the proposed algorithms can be found in Table 9. The LPs were solved using the CPLEX LP solver version 4.0.9 [CPL90]. Computation times were observed on a 200 MHz Sun Enterprise 2. As mentioned in the introduction, our ORTEC datasets contained data describing the transportation problems for four different airplane types, and we are interested in the amount that can be saved if we combine transportation of parts for the different aircrafts. Therefore, we executed our algorithm for every combination of the four different airplane types. We compare the minimum cost of a solution for a given combination of airplane types to the cost of transporting the individual types separately. This gives us the amount of gain we are interested in. The Types column in Table 9 indicates which airplane types are included in a specific problem. The problems are sorted according to increasing size of the constraint matrix. For all problem instances, the number of sites was 6.

The remaining column headers are as follows: the LP solution gives us an upper bound on the amount of gain that can be made by combining the transportation of different types of parts, whereas any feasible solution gives a lower bound. The LP column corresponds to the column generation algorithm as in Section 3.1, MCF corresponds to MCF heuristic as in

Section 2.1, RRLP denotes the combination of the above as in Section 3.4, and Branch and Bound gives information on the performance of the algorithm as explained in Section 4. The Sec. columns give the computation time in seconds. For Branch and Bound, the Gap column gives the size of the duality gap, and the #Nodes column gives the number of nodes in the branch and bound tree that was created. An asterisk (*) indicates that a problem is solved to optimality by the branch and bound algorithm.

The negative gain in the MCF column indicates that the solutions, as returned by the MCF heuristic, are always worse the solutions obtained by combining the optimal solutions to the transportation problem for the involved aircraft types. However, the MCF heuristic, when applied on the excess of a rounded LP solution, gives a very good approximation, and this is really what enables the branch and bound algorithm to produce good results. The essential role of the heuristic is in restoring the feasibility of the solutions as obtained by solving the more sophisticated LP relaxation.

Not all problem instances could be solved to optimality. To find optimal solutions for the remaining cases, we may want to add cutting planes to the LP models. It remains an interesting research topic to develop strong valid inequalities for IPA. Beside this, we considered the problem of finding feasible flows in loading configuration graphs. In Section 5 we explained why it is not easy to generalise combinatorial algorithms for network flow to such a model. It is an interesting topic to find out whether there exist such algorithms, as alternative for solving problems on loading configuration graphs by LP.

Airplane types	Problem size		Upper bound		Lower bound							
			LP		MCF		RRLP		Branch and Bound			
1 2 3 4	$\|L\|$	d	Gain	Sec.	Gain	Sec.	Gain	Sec.	Gain	Sec.	Gap	#Nodes
✓	176	10	0.0%	0.1	−20.0%	0.0	0.0%	0.2	0.0%*	0.2		1
✓	340	10	0.0%	0.2	−29.0%	0.0	0.0%	0.2	0.0%*	0.3		1
✓	198	18	0.0%	0.3	−58.1%	0.0	0.0%	0.3	0.0%*	0.6		1
✓	245	16	0.0%	0.1	−17.5%	0.0	0.0%	0.1	0.0%*	0.2		1
✓✓	510	14	8.8%	0.4	−23.8%	0.0	7.4%	0.5	8.5%	307.5	0.369%	2423
✓ ✓	1190	31	0.8%	1.1	−28.1%	0.1	0.8%	1.2	0.8%*	58.1		199
✓ ✓	1478	28	3.4%	1.9	−33.9%	0.2	2.6%	2.0	3.4%*	74.9		221
✓✓	1626	26	5.9%	2.3	−18.5%	0.2	5.0%	2.3	5.7%	306.0	0.210%	907
✓ ✓	2306	28	5.0%	4.0	−34.7%	0.2	2.9%	4.2	4.3%	304.6	0.808%	531
✓ ✓	2486	26	10.4%	3.5	−17.5%	0.3	10.2%	3.4	10.4%*	6.3		5
✓✓ ✓	2738	32	10.0%	4.7	−28.8%	0.4	9.1%	4.8	9.7%	304.4	0.253%	295
✓✓✓	3007	30	14.2%	4.4	−18.2%	0.4	12.8%	4.5	14.2%	305.3	0.013%	443
✓✓ ✓	4688	41	6.5%	7.5	−25.6%	0.6	5.9%	7.6	6.0%	304.8	0.268%	309
✓ ✓ ✓	6609	41	10.0%	12.4	−25.0%	0.9	9.1%	12.5	9.4%	303.8	0.641%	247
✓✓✓ ✓	7520	45	13.1%	19.1	−23.8%	1.0	12.0%	19.2	12.8%	304.3	0.145%	199

Table 9: Computational Results for IPA.

References

[AMO93] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *Network Flows.* Prentice Hall, Inc., 1993.

[BJN+96] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. Branch-and-price: Column generation for solving huge integer programs. *Opns. Res.* (1996).

[BKOS97] Berg, M. de, Kreveld, M. van, Overmars, M., and Schwarzkopf, O. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.

[CPL90] CPLEX Optimization, Inc. *Using the CPLEX Callable Library*, 1990.

[DDS92] Desrochers, M., Desrosiers, J., and Solomon, M. A new optimization algorithm for the vehicle routing problem with time windows. *Opns. Res.* **40**, 2 (1992), 342–354.

[FF62] Ford, Jr., L. R. and Fulkerson, D. R. *Flows in Networks.* Princeton University Press, 1962.

[GJ79] Garey, M. R. and Johnson, D. S. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[GT88] Goldberg, A. V. and Tarjan, R. E. A new approach to the maximum flow problem. *J. Assoc. Comput. Mach.* **35** (1988), 921–940. Also in Proc. *18th ACM Symp. on Theory of Comp.*, pages 136–146, 1986.

[GT90] Goldberg, A. V. and Tarjan, R. E. Solving minimum cost flow problem by successive approximation. *Mathematics of Opns. Res.* **15** (1990), 430–466.

[HH86] Hall, N. G. and Hochbaum, D. S. A fast approximation algorithm for the multicovering problem. *Discr. Appl. Mathematics* **15** (1986), 35–40.

[MK97] Munier, A. and König, J.-C. A heuristic for a scheduling problem with communication delays. *Opns. Res.* **45**, 1 (1997), 145–147.

[MNU96] Mehlhorn, K., Näher, S., and Uhrig, C. *The LEDA User Manual Version R 3.4.1.* Max-Planck-Institut für Informatik, Saarbrücken, 1996.

[PS82] Papadimitriou, C. H. and Steiglitz, K. *Combinatorial Optimization.* Prentice Hall, Inc., 1982.

[Sol94] Sol, M. Column generation techniques for pickup and delivery problems. Ph.D. thesis, Technische Universiteit Eindhoven, 1994.

[STA97] Shmoys, D. B., Tardos, É., and Aardal, K. Approximation algorithms for facility location problems (extended abstract). In Proc. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing* (El Paso, Texas, 1997), pp. 265–274.

[Thi95] Thienel, S. Abacus—a branch-and-cut system. Ph.D. thesis, Universität zu Köln, 1995.

[Van94] Vanderbeck, F. Decomposition and column generation for integer programs. Ph.D. thesis, Université Catholique de Louvain, 1994.